');

Standard Article

You have full text access to this content

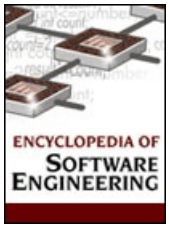
Software Engineering, A Historical Perspective

1. John J. Marciniak

Published Online: 15 JAN 2002

DOI: 10.1002/0471028959.sof321

Book Title



Encyclopedia of Software Engineering

Additional Information

How to Cite

Marciniak, J. J. 2002. Software Engineering, A Historical Perspective. Encyclopedia of Software Engineering. .

Author Information

American Software Profession

Publication History

1. Published Online: 15 JAN 2002

1 Introduction

Software engineering is a relatively young discipline that is evolving at a rapid pace. This pace is advanced by two phenomena: (**1**) the performance capability of computer hardware is advancing rapidly, and (**2**) applications are becoming more complex as the opportunity presented by the attractive performance/price ratio of hardware enables the creation of more diverse and complex applications. This is certainly true in 2001 (at the time of writing), as we have seen the advances in web-based systems and engineering for Internet applications.

2 Definitions

The term “software engineering” was coined at the NATO conference in Garmisch-Partenkirchen in 1968. Since that time, there has been considerable discussion over whether software development is an engineering discipline, and the nature of software engineering itself. Mary Shaw 1990 suggests that it “is not yet a true engineering discipline, but it has the potential to become one” (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof259/sof259.xml?v=1&t=idmlpq1w&s=802acfc20e5fec26da6b89d3905ff300c2648e70)). While most of the discussion has been in academia, we have seen a steady acceptance of results by industry from the research community (e.g., formal methods, advanced design, programming languages). These results have contributed to the advances made in and the discipline of software engineering.

The word *engineering* is an accepted word in today’s practice. In Webster’s *New World Dictionary*, engineering is described as follows:

> (a) the science concerned with putting scientific knowledge to practical uses, divided into different branches, as civil, electrical, mechanical, or chemical engineering (b) the planning, designing, construction, or management of machinery, roads, bridges, buildings, etc. (1982).

The *IEEE Standard Glossary of Software Engineering Terminology* defines software engineering as “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software” (1990).

Thus, software engineers are those who apply engineering principles in the cooperative development and production of systems where software is an intrinsic or dominant part of the overall system. The software engineer brings engineering skills; in terms of software management, selected methods, and the application of tools to the solution of the engineering problem. Others may help in this endeavor: the analyst, who may be engaged in analyzing requirements; the programmer, who is engaged in writing the code that implements the design; and the tester, who is responsible for planning and conducting a test program to determine if the system meets its intended requirements; as well as others. Each of these specializations apply unique software engineering practices. The skill partitioning and organization can assume several dimensions, for example, engineering development, engineering support, and management. In engineering development we have roles of application domain understanding, architecting, design, coding, and testing. In the support role we have quality assurance, configuration management and tool development and support. In the management role we have cost estimation, planning, process management, control or performance management, and risk mitigation. Software engineering, like other engineering disciplines, is a discipline that is devoted to the construction of software systems employing the skills practiced by a variety of people.

2.1 Current Practice

To understand current practice it is useful to examine the evolution of software development practice. In order to do this we employ the concept of characterizing software production environments that dominated over selected periods. Figure 1 depicts this evolution relative to practices that were introduced or popularized over some period of time.

Figure 1. Software engineering genealogy.

2.2 Early Practice

Early computer programming started in the 1940s with the development of the Mark I and ENAIC computers (World Book, 1970). Through the early 1960s, software development was typified by a small programming team environment. Essentially, software was developed by a single programmer, or by a small group of programmers. Programming practice was more or less dictated by the tools available on the computer being programmed. Thus, the availability of the assembler and basic support tools such as a link editor were the tools that were employed and formed the foundation of software development practice. Methods were virtually nonexistent. Flow diagrams were used to construct an overall picture of what the program was intended to accomplish and the programmer coded directly from the flowchart. Cards had to be punched in order to convert source code to machine-readable form. Programs were run in a batch computer environment, where jobs were fed to the machine and executed in sequence, the output fed back in a computer print out.

Management practices were virtually nonexistent, largely a carryover from other management areas. Process was not a concept that was evident at all. Thus, this era was basically defined by the individual programmer. Since many programmers were operating in an environment where their managers had little understanding of what they were doing, and management practices for managing software development were minimal or nonexistent, programming was viewed as a cult with highly individualistic practitioners.

The allocation of effort for software engineering in the early years can be characterized as in Figure 2.

Figure 2. Software development through circa 1980.

2.3 The Project Years

The concept of programming evolved from a programming-in-the-small team environment, to programming in the many. With this larger scale development came an emphasis on project management of software development (see Fig. 3).

Figure 3. Allocation of effort, early years.

As software systems became more complex, for example, the development of early military systems such as the Semi-Automated Ground Environment (SAGE) developed by the United States (U.S.) Department of Defense (DOD) 1988 to monitor the airspace surrounding its borders, it became clear that developing systems in a single programmer environment of this first generation was inadequate. SAGE involved an application program of 100,000 instructions with a support system of over 100 million instructions and was implemented on a computer with 58,000 vacuum tubes. (Augustine, 1983) It was this increasingly wider use of software in complex and critical areas that provided the impetus for better management and engineering (or programming techniques depending on the personal point of view at the time) for developing software.

From a product development—or production—point of view, programming was advanced through the use of high order languages (HOLs), the most common of which were FORTRAN and COBOL, at least in the United States. Abstraction techniques such as "top-down" structured design were introduced in an attempt to provide, to the programmer, methods that allowed better analysis of requirements and formulations of design. Programmer tools were still pretty basic, relegated to debuggers and other tools found in the programmer toolbox. Testing was largely an art more often than not forgotten until late in the development, and then mostly a functional demonstration that the system worked so the client would accept it.

Project management was now seen as a necessity to develop software systems properly. The programmer was no longer a single person or small group operating entirely on their own. Management methods, although still basic, were emerging. Cost estimation and tracking was virtually nonexistent. Tracking of development was accomplished principally by line-of-code projection and reporting of actual lines of code that were written. Management of the evolution of several cooperating components required configuration management. Managed development of many related activities demanded planning tools with techniques such as PERT (Marciniak and Reifer, 1990).

If the early part of this era was best known for programming techniques, the latter part is best known for an emphasis on quality (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof264/sof264.xml?v=1&t=idmlpq2j&s=9636534f7c00720efbb162b8006a320b6b4b2836)). Project management practices were dramatically changing. There were many reasons for this depending on one's point of view. In a commercial environment, the opportunity to develop and sell programming services for financial services such as payroll production was a motivation. In the U.S. DOD the amount of software was growing, and for the first time, was appearing in weapons systems such as missiles and aircraft. In the National Aeronautics and Space Agency (NASA), the software dramatically jumped in the development of the Space Shuttle, where initial estimates paled compared to the resulting 25 million lines of code (Schlender, 1989).

In the United States the most important initiative was the development of the Ada language (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof006/sof006.xml?v=1&t=idmlpq2o&s=d59c6863b4415fc58de95f5dea1775b84298fa3e)). While still somewhat controversial, this development focused enormous attention and resource to software engineering issues such as the use of strong typing in HOLs and information hiding. Ada created an international following, and today still has a strong following. As the importance of software grew, both in the government and commercial sectors, national efforts were organized to focus resources on improving the productivity and quality of software production.

2.4 The Process Years

The process concept developed during the 1970s–early 1990s (see Fig. 4).

Figure 4. The process years.

The Japanese, in 1982, described the fifth generation computer project. This project, had as its intention the development of a computing environment based on knowledge-based systems and computers that directly executed advanced or fifth generation languages such as Prolog (Cusumano, 1991b). The Japanese, however, are best known for their implementation of the "software factory." This term, initially conceptualized in the United States at Systems Development Corporation, was implemented in Japan during the 1980s and set the standard for producing software in a highly disciplined environment (Cusumano, 1991a). The Japanese software factory is grounded in two characteristics: the use of highly trained and concentrated development teams, and the application of metrics to measure process improvement and product quality and improve the factory based on these measures. While software metrics had an early beginning in the late 1970s the Japanese made it a cornerstone of the software factory concept.

In 1982 the ALVEY project was organized in the United Kingdom when the ALVEY committee recommended a broad program in research in information technology. The program was spurred by the Japanese initiative. The five year program involved major IT firms such as BT, and was funded on a 50/50 cost sharing basis. In the five years 198 projects were carried out involving 2500 researchers.

The European community formed the European Strategic Program for Research and Development (ESPRIT) in 1984 to develop software technology to cope with this expending commercial base in information technology, as well as in microelectronics, advanced information processing, and computer integrated manufacturing. Thirty-eight projects were launched in the pilot phase. As of mid-1992 a total of 721 distinctive results had been achieved. Examples of ESPRIT programs are the Portable Common Tool Environment (PCTE), RAISE, an enhancement to the VDM method, BOOTSTRAP, which helped map the Software Engineering Institute (SEI) Capability Maturity Model (CMM) onto the International Organization for Standardization (ISO) 9000 series quality standards, and MUSIC, focusing on the development of metrics. ESPRIT has been asorbed in the IST Programme (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof172/sof172.xml?v=1&t=idmlpq32&s=aab46430fb5a7f89c6d224c8e85ff67e1c42038d)). The U.S. DOD organized the Software Engineering Institute (SEI) in 1984, hosted at the Carnegie Mellon University. The objective of this Federally Funded Research and Development Center (FFRDC) is to advance software engineering practice and improve quality (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof317/sof317.xml?v=1&t=idmlpq34&s=4cd2facc0c13ea520cb5297ba5fa7786f79e06d7)).

The advent of the software factory brought new practice and attention to the infrastructure of software development, and set the stage for the focus on the process of software development. With measurement other practices began to emerge. In order to measure productivity, the basic unit of work needed definition. Many attempts were made to define a "line of code" so that the basis for comparison between systems was clear. Productivity data that were available provided the early means to develop and implement cost estimation practices. Error data provided the means for implementing reliability mechanisms that had been under development, but had not been introduced into practice (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof327/sof327.xml?v=1&t=idmlpq36&s=a6133c6ebc0862ab700331b23fe6725802bc3f29) and (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof282/sof282.xml?v=1&t=idmlpq37&s=61fc535a6548115c3e2bc284ef85cf38ff1e1f4a)).

Perhaps the most significant happening in this time was the concentration on process improvement programs. Initially conceived in the United States by Watts Humphrey while at IBM, then later at the SEI, its impetus can be attributed to the gains that the Japanese were making in managing the software development process in their software factories. Now codified by method and practice by the SEI, the process improvement program is an extension of the principles of total quality management (TQM) customized for software development. The goals of the program are to instill a process improvement program in industrial practice and to establish the means to quantify and assess the attainment of the program (Humphrey, 1989) (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof589/sof589.xml?v=1&t=idmlpq3a&s=dccf1c805fe7e2d06bb3e7629d33b56574f27afd), (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof359/sof359.xml?v=1&t=idmlpq3c&s=cace04cdbfb811ad473933a6d956d1266a4971e4), and (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof360/sof360.xml?v=1&t=idmlpq3d&s=3ed08321fb140dc696335a019b5871e6e06ccb05)).

The quality emphasis of this generation saw the development and implementation of new standards of practice. The first management process/life cycle standard was Federal Information Processing Standard (FIPS) 38, Guidelines for Documentation of Computer Programs & Automated Data Systems. Published in 1976 it is now obsolete, however, it provided the basis for other management and development standards. In 1988 the United States published DOD-STD-2167A, Standard for Defense System Software Development. It provided a common life-cycle management process for the development of software in systems. While it received criticism, mainly in implementation issues and the reliance on a waterfall life-cycle model, it did provide a basis for standard development practice for U.S. DOD weapon systems. It has since been replaced in usage by ISO/IEC 12207 (IEEE/EIA, 1966). The IEEE was developing an active standards program (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof319/sof319.xml?v=1&t=idmlpq3i&s=46339230610cebba88f943c0d0fd002fae788d70)) as well as the International Organization for Standardization (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof164/sof164.xml?v=1&t=idmlpq3j&s=89cf1d2ec667685495ed45477c6e9301c6d19379)).

2.4.1 *Methods*

Perhaps the most important method that has emerged since 1990 was object-oriented development. Originally supported by the object-oriented language and environment Smalltalk, it is joined by Eiffel and C++ (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof030/sof030.xml?v=1&t=idmlpq3m&s=b467a129e0c4fc6e21bcabde3a1b3fb425757176) and (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof304/sof304.xml?v=1&t=idmlpq3n&s=7c0ab466434268ba9146188dd9ab550e3eab7545)).

2.4.2 *Tools*

In the process years we saw a number of tool or computer-aided software engineering (CASE) companies arise with new offerings. Examples are CADRE, which made a product called *Teamwork*; Interactive Development Environments (IDE), which made *Software through Pictures*; and Mark V, which made *Object-maker*. The explosion in tools matched the capabilities of lower cost, higher performance workstations that allowed even more sophisticated tools. Ascent Logic Corporation offered a tool capability called RDD100, which is based on the technology and method of SYSREM (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof330/sof330.xml?v=1&t=idmlpq3r&s=f96fa144a2ecbe19a3f4e1335872ceac703b036c)). Virtual Software factory made a metatool called VSF, which affords the rapid development and automation of methods to create custom tools. McCabe Associates still markets a number of code analyzers, based on techniques such as complexity analysis, that help in the testing process.

Early software engineers were trained as programmers. Since there were no academic curricula for software engineering, their training was characterized as “on the job” supplemented by industrial and commercial programming courses. Programmers stemmed from a variety of educational backgrounds. In 1976 Peter Freeman, then with the University of California at Irvine, and Anthony Wasserman, then with the University of California at San Francisco, organized a workshop on software engineering education (see (http://onlinelibrary.wiley.com/store/10.1002/0471028959.sof321/asset//sof098/sof098.xml?v=1&t=idmlpq3x&s=3aeb4354e0f853c2d5fad2a87ebc6652e38aae79)). As the era proceeded, computer science programs were organized in many academic institutions. These were mainly found in the mathematics departments of universities and colleges. The first software engineering curriculum was offered by Texas Christian University, in 1978. By the end of the seventies this software engineering curricula was taking hold in a number of institutions. Since the curricula were at the graduate level, the major source for software engineers was still drawn from a variety of computer science, engineering, and mathematics programs.

The allocation of software engineering effort in the process era is depicted in Figure 5 (Boehm, 1981). The shaded area shown in the testing phases pays homage to the fact that sufficient resources were not allocated to testing, however, in typical developments the testing phase increased in resource requirements.

Figure 5. Allocation of effort, process years.

This process era, then, is best characterized by the following salient points:

Definition and organization of national initiatives to promote the use and development of software (e.g., ESPRIT)

Development of process improvement programs (e.g., SEI CMM)

Codification of management practice in standards (e.g., DOD-STD-2167A)

Definition and implementation of software engineering curricula

2.5 The Production Era

While it would be presumptuous to state what will happen in the future, it is possible to report on the events that have occurred since the mid 1990s. We term this era the *production years* because of the interest in and capability to produce varied systems. This era is somewhat characterized by the develolpment of Web-based systems. The production era has been captured in this encyclopedia through the update of existing articles and introduction of new ones.

- *Production*. From a production perspective the most change, or novelty, has been with Web-based development. New articles include
    - Architecture
    - Component-based systems

    - Design patterns
    - Engineering Web Applications with Java
    - Extreme Programming
    - Java
    - Middleware
    - Omg/Corba
    - Reading techniques
    - Scripting languages
    - Software cost reduction method
    - Unified Modeling Language
    - Unified Process
    - Visual programming

- *Process*. Process certainly had become a big-time activity. Initially with the SW-CMM, the field has expanded with a number of process models across areas such as "personal," "team," "systems engineering," and "acquisition." Below are the encyclopedia articles that expand this area.
    - Capability Maturity Model for Software
    - Cmmi
    - Integrated Process Models
    - ISO 9000
    - ISO/IEC 15504 and Spice
    - People CMM
    - Personal Software Process
    - Software Acquisition Capability Maturity Model
    - Software process assessments
    - Statistical process control of software
    - Team software process

- *Project*. On the project manageet side certainly the "process" has made a major difference in how people approach project management. There is also more interest in, and application of risk management techniques. Articles of interest in this area include
    - Earned value
    - Function points
    - Software engineering and litigation
    - Peer reviews
    - Project management
    - Requirements traceability
    - Requirements management
    - Resource estimation
    - Risk management
    - Risk management in software development
    - Safety
    - Test management and organization

- Test automation

Bibliography

N. R. Augustine, Augustine Laws, American Institute for Aeronautics & Astronautics, New York, 1983.

B. W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981, Chapters 6 and 7.

M. A. Cusumano, Japan's Software Factories, Oxford University Press, New York, 1991a, p. 7.

M. A. Cusumano, Japan's Software Factories, Oxford University Press, New York, 1991b, pp. 41–416.

Department of Defense, Standard for Defense System Software Development, Department of Defense Standard 2167A, DoD, Washington, DC, 1988.

W. S. Humphrey, Managing the Software Process, Addison-Wesley, Reading, MA, 1989.

IEEE, Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, IEEE, New York, 1990.

ISO/IEC, Standard for Information Technology, ISO/IEC 12207, IEEE/EIA 12207.0-1966, ISO/IEC, Geneva, 1966.

J. J. Marciniak and D. J. Reifer, Software Acquisition Management, Wiley, New York, 1990, pg. 9.

*B. R. Schlender, How to Break the Software Logjam, Fortune **120**, 100–112 (September 25, 1989).*
Web of Science® Times Cited: 1 (http://onlinelibrary.wiley.com/resolve/reference/ISI?id=A1989AN69800008)

M. Shaw, Prospects for an Engineering Discipline of Software, IEEE Software, pp. 15–24 (November 1990).

Webster's, New World Dictionary, 2nd College edition, Simon & Schuster, New York, 1982.

World Book Encyclopedia, Computer, World Book, Chicago, IL, 1970, Vol. 4, p. 744.